



TITLE:

音声認識を用いた日本語による数式インタフェース (数学解析の計算機上での理論的展開とその遂行可能性)

AUTHOR(S):

前田, 秋吐; 鈴木, 昌和

CITATION:

前田, 秋吐 ...[et al]. 音声認識を用いた日本語による数式インタフェース (数学解析の計算機上での理論的展開とその遂行可能性). 数理解析研究所講究録 2002, 1286: 170-177

ISSUE DATE:

2002-09

URL:

<http://hdl.handle.net/2433/42469>

RIGHT:

音声認識を用いた日本語による数式インタフェース

前田 秋吐[†] 鈴木 昌和[†]

[†] 九州大学大学院数理学数理学府数理学専攻 〒812-8581 福岡県福岡市東区箱崎 6-10-1

E-mail:suzuki@math.kyushu-u.ac.jp

あらまし エディタなどへの数式入力支援のために、キーボードによらない音声による数式入力インタフェースの研究を行っている。音声認識には、話者不特定の認識ができる AmiVoice を用いている。認識の処理には、まず形態素解析を行う。数式の読み上げで、息継ぎするところを工夫することにより、さまざまな構造の数式が認識できるようにした。次に、日本語読みに適した数式の文法を作成し、それを用いた構文解析をおこなって数式構造を構築している。数式の認識対象範囲は、おもに、中学、高校で取り扱う数式を想定している。さまざまな読み方を認識するように、表現の幅をもたせている。認識処理の手法について説明し、デモを行う。

キーワード 音声認識、形態素解析、構文解析

Mathematical Input Interface used Japanese and Voice recognition

Akito Maeda[†] Masakazu Suzuki[†]

[†] Graduate School of Mathematics, Kyushu University 6-10-1 Hakozaki, Higashi-ku, Fukuoka, 812-8581 Japan

E-mail:suzuki@math.kyushu-u.ac.jp

Abstract To support for mathematical input about edit application and so on, We are studying about mathematical interface for voice not to depend on keyboard. We use AmiVoice application in Voice input. It is enabled to recognize not to depend on person's voice. At first, we execute morphological analyser in recognition disposal. A speaker devises breath point in reading out of math equation. And interface can recognize verious structure of math equation. Next, we make mathematical grammar suited japanese reading. And we execute syntax analysis them. And math structure generated. This interface are supported to math equation used to junior or high school about math recognition range. We seted up expression range to recognize verious reading procedure. We explain about method of recognition management, and We demonstrate voice interface.

Keywords Voice recognition, morphological analyser, syntax analysis

1 はじめに

現在のネットワークやコンピュータの加速的な普及に伴い、あらゆる分野において様々な可能性が生まれている。数学においても例外ではない。単なる計算だけに留まらず、数式をコンピュータ上で取り扱うための多くの環境が開発され、用意されてきた。特に、数学の研究や教育の分野においては、目覚ましい発展をしている。例えば、数式の計算、グラフ描画などを円滑に行える Mathematica や Maple, MathLab といった数式処理システムがある。学術情報データベースに接続した電子ボードなど、数学の授業や講義での新しいスタイルも現れてきた。T_EX や MathML といった各種数式フォーマット、各種ワープロに付属の数式入力機能などは、個人のコンピュータに導入されるまでになってきている。

Mathematica や各種ワープロに付属の数式入力機能などは、直感的で分かりやすい。しかし、メニューボタンなどによる入力操作を、繰り返さなければならない。その煩わしさのため、スムーズな入力を妨げ、ユーザーの思考の流れを中断させる可能性も高い。T_EX は数式のスムーズな入力が行える。しかし、使いこなせるようになるまでには、ある程度の修練が必要とされる。入力された数式の意味を一目見ただけでは、把握できない点もある。

誰でもオンライン上で数式を扱えるようにするためには、より快適な操作環境が要求される。

現在、言葉での音声による入力を行うアプリケーションが普及してきており、徐々にではあるが、コンピュータ購入時からでも、標準で装備されるようになってきた。一般に、エディタ画面上に、音声による文字の入力を行ったり、音声によるコマンド操作や、入力を行ったりすることができるようになってきている。また、電話会社のなかには、音声コマンド入力や音声入力による情報検索案内などの各種サービスも、提供しはじめているところがある。

しかし、いまだに数式に関しては、音声による入力インタフェースが、存在していない。

そこで、今回の研究では、ユーザーの、音声による数式での入力インタフェース開発を行うことにした。ハード面で、音声認識は、入力を受け付けるマイクと、入力した内容を確認するための表示機能さえあれば、実現される。ソフト面では、煩雑なボタンの操作や切り替え、キーボードのタイピング、時間を大幅に軽減することができる。これにより、例えば、携帯端末など小型機器への音声による数式入力が可能になる。

日本語で数式を読み上げるときに必要なのが、数式に関連する単語を一語ずつ正確に認識することである。そのためには、数式に対する読み方の規則性についても定めなければならない。日本語での数式の読み上げには、統一的な読み方が存在していない。その人、個人によっ

て、読み方に、かなりの差が生じる。そこでまず、日本語数式の多様な読み方に対して、統一的な規則性をつくることにした。しかし、できるだけ汎用的な読み方を行えるようにするため、読み方に表現の幅も持たせてある。

日本語読み数式の音声による入力は、入力操作の予備知識を必要としない。また入力過程に関しては、入力して受理された数式をリアルタイムにエディタ画面上に表示するようにしている。一目で分かるという利点を持ち、日本語読み数式音声入力インタフェースとして大変利用しやすい仕上がりとなっている。

話者により、単語の発声の仕方は様々である。正確な認識を 100 パーセント行えるかどうかは、今後の多くの実験にゆだねられる。あらゆる話者に対応し得る数式音声認識システムが開発されたことは、入力の実験にも、安定した認識結果をもたらしてくれるものと思われる。

今回、システムで用いる日本語読み数式音声入力処理に至るまでの構築過程を述べていく。

2 音声認識用の辞書とルールグラムの組み込み

現在、日本語で音声認識のできる音声認識エンジンが、複数の企業から製品化されている。エンジンが、各個人の話し方にとらわれずに、学習を必要としなければ、すぐにでも利用することができる。そこで、話者不特定の認識が可能なアドバンスト・メディア社製の AmiVoice アプリケーションを利用することにした¹。AmiVoice には、さまざまな機能が用意されている。ここでは、日本語読み数式音声入力に必要な機能について取り上げる。また、AmiVoice から、今回の研究に必要なファイルなどの組み込みについて説明する。

2.1 ユーザー辞書の組み込み

音声認識での数式用の辞書は、math という名前で作成している。math を、AmiVoice エンジン初期化と同時に、指定することで、認識が可能となる。

2.2 日本語読み数式用ルールグラム

ルールグラムは、AmiVoice 準拠の、Java Speech Grammar Format (JSGF) 形式で記述する。AmiVoice エンジン初期化と同時に、指定することで、認識が可能となる。今回作成した、グラムファイルは、以下のようになる(一部のみ掲載)。グラム名は、grammar_math である。“<”と“>”で囲まれた部分はルール名を表す。“{”と“}”で囲まれた部分は、タグ名で、筆記表記²とともに書かれる。書き方のルールは、正規表現に対応している。ルール名は、非終端記号に対応しており、筆記表記は、終端記号に対応している。

¹ 正確には、AmiVoice SDK version 4.0 ソフトウェアで、アドバンスト・メディア株式会社から音声認識開発ツールキットとして、提供していた。

² これについては第 4 章の部分で詳述。

| | |
|--|---|
| 分数 | |
| 「[式] 分の [式]」 | $\frac{[\text{式}]}{[\text{式}]}$ |
| 「[式] 分の [式], 分の, [式] 分の [式]」 | $\frac{\frac{[\text{式}]}{[\text{式}]}}{[\text{式}]}$ |
| 「分数 分子 [長い式] 分母 [長い式]」 もしくは 「分数 分母 [長い式] 分子 [長い式]」 | $\frac{[\text{長い式}]}{[\text{長い式}]}$ |
| 「分数始まり 分子, [式], [式], ... 分母, [式], [式], ..., 分数終わり」 | $\frac{[\text{式}][\text{式}] \cdots}{[\text{式}][\text{式}] \cdots}$ |
| 「三 分の 四」 | $\frac{4}{3}$ |
| 「括弧 セット 分の 一 括弧閉じ, 分の, 一」 | $\frac{1}{\left(\frac{1}{x}\right)}$ |
| 「一 足す エックス 分の サインシータ」 | $\frac{\sin \theta}{1+x}$ |
| 「二 ルート ワイ 分の 一」 | $\frac{1}{2\sqrt{y}}$ |

以下, おもな, 数式の読み方について, 代表的な例をあげる。

例 3.1 (積分) $\int_0^{\frac{1}{2}} \frac{x^4}{1-x^2} dx.$

「イントの 零から 二分の一 まで, マイナス エックスの二乗 分の エックスの 四乗, ディー エックス」。

例 3.2 (解の公式) $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

「エックス イコール, ニ エー 分の マイナス ビー プラ マイ ルート ビー の 二乗 マイナス 四 エー シー」。

例 3.3 (加算和) $\sum_{k=1}^n a_k = a_1 + a_2 + \cdots + a_n.$

「サム の けー は 一 から エヌ まで, エー の ケー, イコール, エー の 一, プラス エー の ニ, プラス シードツ プラス エー の エヌ」。

例 3.4 (数字, 小数点) $500 \times 0.9613 = 480.65$

「五百 掛ける 零 点 九 六 一 三 イコール 四百八十 点 六 五」。

例 3.5 (ベクトル, ベクトル成分) $\overrightarrow{OA} = (a_1, a_2).$

「ベクトル ラージオー ラージエー, イコール, 括弧始まり, エー の 一, コンマ, エー の ニ, 括弧終わり」。

例 3.6 (共役な複素数) $\overline{\left(\frac{z_1}{z_2}\right)}.$

「オーバーライン 括弧 セットの 二 分の ゼットの 一 括弧閉じ」。

例 3.7 (集合記号) $\overline{A \cap B} = \{1, 3, 6\}.$

「オーバーライン ラージエー キャップ ラージビー, イコール, 中括弧 一 カンマ 三 カンマ 六 中括弧閉じ」。

例 3.8 (対数) $y = \log_{\frac{1}{2}} \frac{1}{x^2}.$

「ワイ イコール, ログ 底 二分の一, エックスの二乗 分の 一」。

例 3.9 (対数) $x^{\log_{10} x^8}.$

「エックス 上付き始まり, ログ 底 十, エックスの三乗, 上付き終わり」。

例 3.10 (極限) $\lim_{h \rightarrow 0} \frac{\frac{1}{x+h} - \frac{1}{x}}{h}.$

「リミットの エイチ ライトアロウ 零, 分数始まり 分子 エックス プラス エイチ 分の 一, マイナス, エックス分の 一, 分母 エイチ, 分数終わり」。

例 3.11 (合同) $\triangle ABC \equiv \triangle PQR.$

「三角形 ラージエー ラージビー ラージシー 合同 三角形 ラージビー ラージキュウ ラージアール」。

4 日本語読み数式の形態素解析

日本語の形態素解析の処理には, 数多くの言語知識が必要とされる (参考文献 [2] を参照) が, 今回, AmiVoice 音声認識エンジンを利用することにより, その作業を大幅に軽減させることが可能となった。

4.1 形態素解析のための音声データ取り出し

なんらかの音声を認識させたとき, AmiVoice 音声認識エンジンから, 生の音声データとして, 筆記表記, 口述表記, タグ表記, タイムタグ表記, ルール名表記が, それぞれ, “|” で区切られた文字の羅列として返される。各表記列を, 1 単語ずつに分けて, その 5 種を対応させる。今回, 口述表記とルール名表記は使用していない。

$$\frac{2ax}{x^2 - ax + 1}$$

とすると, AmiVoice から返される生の音声データは次のようになる。

筆記表記列

x | of | 2 | 乗 | - | a | x | + | 1 | 分の | 2 | a | x

口述表記列

えつくす | の | に | じょう | まいなす | えー | えつくす | ぶらす | いち | ぶんの | に | えー | えつくす

タグ表記列

OneChar | Of | Number | Exponent | OneChar | OneChar
| OneChar | OneChar | Number | FractionTag | Number |
OneChar | OneChar

タイムタグ表記列

0 | 230 | 450 | 730 | 980 | 1200 | 1450 | 1700 | 1970 | 2500 |
2910 | 3100 | 3320

このようなデータを格納するために, 文字列構造体の双方向リストを作成し, 音声データ情報を格納する。それぞれの表記情報を文字列ポインタに格納している。

4.2 形態素解析後の情報の追加

形態素解析したリストに対して、さらに、構文解析を行いやすくするための情報を付け加えていく。追加する情報は、以下の2点である。

- タイムアウトタグから、数式の区切りをつけ、リストに区切り記号を追加する
- リストの最後尾に解析の終わりを記す記号を追加する

タイムアウトタグによる数式の区切り

まず、リスト上の前の単語と、後の単語に対して、タイムアウトタグの差をとる。その差が、500 msec 以上あれば、数式を区切る。区切りの記号をその単語間に挿入する。区切りには、一息で発声した数式の最後の単語の後に、筆記表記、“EndBreath”，タグ表記，“EndBreath”，タイムタグ表記，“-”を、数式の最初の単語の前に、筆記表記，“StartBreath”，タグ表記，“StartBreath”，タイムタグ表記，“-”を記号として用いる(表1, p5を参照)。

リストの終わりの記号

終わりの記号は、筆記表記，“\$”，タグ表記，“\$”，タイムタグ表記，“-”として追加した。この終わりの記号は、構文解析するとき、入力列の最後を認識させるために必要である(表1, p5を参照)。

表1: 形態素解析した表

| 筆記表記 | タグ表記 | タイムタグ表記 |
|-------------|-------------|---------|
| StartBreath | StartBreath | - |
| x | OneChar | 0 |
| of | Of | 230 |
| 2 | Number | 450 |
| 乗 | Exponent | 730 |
| - | OneChar | 980 |
| a | OneChar | 1200 |
| x | OneChar | 1450 |
| + | OneChar | 1700 |
| 1 | Number | 1970 |
| EndBreath | EndBreath | - |
| 分の | FractionTag | 2500 |
| StartBreath | StartBreath | - |
| 2 | Number | 2910 |
| a | OneChar | 3100 |
| x | OneChar | 3320 |
| EndBreath | EndBreath | - |
| \$ | \$ | - |

5 日本語読み数式の文法作成

日本語読み数式文法は、直接、数式入力文字列の構文解析を行うための必要性から導入した。文法が完成して

初めて、構文解析譜を作成することができる。できるだけ日本語での数式の読み方に対応させるために以下で取り上げる独自の生成規則⁴から構成した。しかし、長い数式文字列に対しては、それだけでは不足する。TeXの数式フォーマットにみられるような、読み方も取り入れて、多様な読み方に対応させている。

5.1 日本語読み数式の文法

$G = \{N, T, P, \text{Math}\}$.

$N = \{$

Math, MathUnit, MathEquation, MathEquationStart, MostLong, Fraction, MostLongFraction, Numerator, Denominator, LongNumerator, LongDenominator, Root, RootBase, IndexRoot, MostLongRoot, MathSub, IndexRootBase, DirectRoot, SubSuper, LongSubSuper, SubSuper, MathSubBase, MathSubSuper, MathSubSuperBase, SubScript, SuperScript, LongSuperScript, LongSubScript, MathOver, MathOverBase, OverScript, MathParenthesis, MathParenthesisBase, LeftParenthesis, RightParenthesis

$\}$.

5.2 日本語数式読み生成規則の作成

- 数式の、日本語での一般的な読み方をモデルとして生成規則の形にまとめる。
- 生成規則を、LL(1) 文法に適するような形になおしていく。
- 左再帰性と後戻りをあらかじめ取り除いておく。
- 冗長な生成規則をなくして規則数を減らしていく。
- 最終的に、生成規則総数は23個となる。

$P = \{$

(1) Math \rightarrow MathUnit*

(2) MathUnit

\rightarrow MathEquationStart | MathSub | MathParenthesis | MathSubSuper | MathOver | DirectRoot | LongFraction | MostLong

(3) MathEquationStart

\rightarrow MathEquation (Fraction | IndexRoot | SubSuper | LongSubSuper | Empty)

(4) LongSubSuper

\rightarrow SubScriptStart MathEquation SubScriptEnd (SuperScriptStart MathEquation SuperScriptEnd | Empty) | SuperScriptStart MathEquation SuperScriptEnd (SubScriptStart MathEquation SubScriptEnd | Empty)

(5) SubSuper \rightarrow Of MathEquation(Exponent | Empty)

(6) LongFraction

\rightarrow LongFractionStart (DenominatorTag MathEquation NumeratorTag MathEquation | NumeratorTag MathEquation DenominatorTag MathEquation) LongFractionEnd

⁴日本語読み数式の文法を作成するにあたり、旧鈴木研究室に在籍されていた岡村博文氏の作成した文法を参考にした(参考文献[5]を参照)⁵。

- (7) **Fraction** \rightarrow **FractionTag MathEquation**
- (8) **IndexRoot** \rightarrow **IndexRootBase DirectRoot**
- (9) **IndexRootBase** \rightarrow **IndexRootTag | Empty**
- (10) **DirectRoot** \rightarrow **RootStart MathEquation**
- (11) **MathSub**
 \rightarrow **MathSubStart (MathEquation | Empty)**
- (12) **MathSubSuper**
 \rightarrow **MathSubSuperBase MathEquation (From (MathEquation To | Empty) | To (MathEquation From | Empty) | Empty)**
- (13) **MathOver** \rightarrow **MathOverStart MathEquation**
- (14) **MathEquation**
 \rightarrow **OneChar | StartBreath Math EndBreath**
- (15) **MathParenthesis**
 \rightarrow **LeftParenthesis MathParenthesisBase RightParenthesis**
- (16) **MathParenthesisBase** \rightarrow **Math**
- (17) **MostLong**
 \rightarrow **MostLongFraction | MostLongRoot | MostLongMathOver | MostLongMathSub**
- (18) **MostLongFraction**
 \rightarrow **MostLongFractionTag LongDenominator LongNumerator**
- (19) **LongDenominator** \rightarrow **DenominatorStart MathEquation DenominatorEnd**
- (20) **LongNumerator** \rightarrow **NumeratorStart MathEquation NumeratorEnd**
- (21) **MostLongRoot** \rightarrow **LongRootStart MathEquation LongRootEnd**
- (22) **MostLongMathOver**
 \rightarrow **LongMathOverStart MathEquation LongMathOverEnd**
- (23) **MostLongMathSub** \rightarrow **LongMathSubStart MathEquation LongMathSubEnd**

}.

5.3 日本語読み数式文法の LL(1) 文法との妥当性

この文法が, LL(1) 文法かどうかを確かめる.

First(MostLongMathSub) = { LongMatSubStart }

First(MostLongMathOver)
 = { LongMatOverStart }

First(MostLongRoot) = { LongRootStart }

First(LongNumerator) = { NumeratorStart }

First(LongDenominator) = { DenominatorStart }

First(MostLongFraction)
 = { MostLongFractionTag }

First(MostLong)
 = { MostLongFractionTag, LongRootStart, LongMatOverStart, LongMatSubStart }

First(MathParenthesisBase) = **First(Math)**

First(MathParenthesis) = { LeftParenthesis }

First(MathEquation) = { OneChar, StartBreath }

First(MathOver) = { MathOverStart }

First(MathSubSuper) = { MathSubSuperBase }

First(MathSub) = { MathSubStart }

First(DirectRoot) = { RootStart }

First(IndexRoot) = { IndexRootTag, RootStart }

First(Fraction) = { FractionTag }

First(LongFraction) = { LongFractionStart }

First(SubSuper) = { Of }

First(LongSubSuper) = { SubScriptStart }

First(MathEquationStart)
 = { OneChar, StartBreath }

First(MathUnit)
 = { OneChar, StartBreath, RootStart, IndexRootTag, LongFractionStart, MathSubStart, MathOverStart, LeftParenthesis, MostLongFractionTag, LongRootStart, LongMatOverStart, LongMatSubStart }

First(Math)
 = { OneChar, StartBreath, RootStart, IndexRootTag, MathSubStart, MathOverStart, LeftParenthesis }

Follow(IndexRoot) = { RootStart }

よって

Director(Math, MathUnit)

= { OneChar, StartBreath, RootStart, IndexRootTag, LongFractionStart, MathSubStart, MathOverStart, LeftParenthesis, MostLongFractionTag, LongRootStart, LongMatOverStart, LongMatSubStart }

Director(MathUnit, MathEquationStart)

= { OneChar, StartBreath }

Director(MathUnit, MathSub)

= { MathSubStart }

Director(MathUnit, MathParenthesis)

= { LeftParenthesis }

Director(MathUnit, MathSubSuper)

= { MathSubSuperBase }

Director(MathUnit, MathOver)

= { MathOverStart }

Director(MathUnit, DirectRoot)

= { RootStart }

Director(MathUnit, LongFraction)

= { LongFractionStart }

Director(MathUnit, MostLong)

= { MostLongFractionTag, LongRootStart, LongMatOverStart, LongMatSubStart }

Director(MathEquationStart, MathEquation (Fraction | IndexRoot | MathSubSuper | SubSuper | LongSubSuper | Empty))

= { OneChar, StartBreath }

Director(Fraction, FractionTag MathEquation)

= { FractionTag }

Director(SubSuper, Of MathEquation (Exponent | Empty))
 = { Of }

Director(LongSubSuper, SubScriptStart MathEquation SuperScriptEnd (SuperScriptStart MathEquation SuperScriptEnd | Empty) | SuperScriptStart MathEquation SuperScriptEnd (SubScriptStart MathEquation SuperScriptEnd | Empty))
 = { SubScriptStart }

Director(LongFraction, LongFractionStart (DenominatorTag MathEquation NumeratorTag MathEquation | NumeratorTag MathEquation DenominatorTag MathEquation) LongFractionEnd)
 = { LongFractionStart }

```

Director( IndexRoot, IndexRootBase DirectRoot )
= { IndexRootTag, RootStart }
Director( IndexRootBase, IndexRootTag | Empty )
= { IndexRootTag, RootStart }
Director ( DirectRoot, RootStart MathEquation )
= { RootStart }
Director ( MathSub, MathSubStart ( MathEquation |
Empty ) )
= { MathSubStart }
Director ( MathSubSuper, MathSubSuperBase MathEquation ( From ( MathEquation To | Empty ) | To ( MathEquation From | Empty ) | Empty ) )
= { MathSubSuperBase, OneChar, StartBreath }
Director ( MathOver, MathOverStart MathEquation )
= { MathOverStart }
Director ( MathParenthesis, LeftParenthesis MathParenthesisBase RightParenthesis )
= { LeftParenthesis }
Director ( MathParenthesisBase, Math )
= { OneChar, StartBreath, RootStart, IndexRootTag, MathSubStart, MathOverStart, LeftParenthesis }
{
  Director( MostLong, MostLongFraction )
  = { MostLongFractionTag }
  Director( MostLong, MostLongRoot )
  = { LongRootStart }
  Director( MostLong, MostLongMathOver )
  = { LongMathOverStart }
  Director( MostLong, MostLongMathSub )
  = { LongMathSubStart }
}
Director ( MostLongFraction, MostLongFractionTag LongDenominator LongNumerator )
= { MostLongFractionTag }
Director ( LongDenominator, DenominatorStart MathEquation DenominatorEnd )
= { DenominatorStart }
Director ( LongNumerator, NumeratorStart MathEquation NumeratorEnd )
= { NumeratorStart }
Director ( MostLongRoot, LongRootStart MathEquation LongRootEnd )
= { LongRootStart }
Director ( MostLongMathOver, LongMathOverStart MathEquation LongMathOverEnd )
= { LongMathOverStart }
Director ( MostLongMathSub, LongMathSubStart MathEquation LongMathSubEnd )
= { LongRootStart }

```

以上より、 $\forall A \in N$ と $\forall A$ を左辺に持つすべての生成規則に対して、Director に共通部分が存在しないので、この文法が LL(1) 文法であることが示された⁶。

6 日本語読み数式の構文解析

今回の日本語読み数式文法は、LL(1) 文法であると確認したあと、アルゴリズムの比較的組み立てやすい LL(1) 構文解析 (参考文献 [1] を参照) を用いることにした。

⁶ここでは、具体的に、一つ一つ Director を求めて確かめたが、始めの文法入力をするだけで、自動的に、LL(1) 文法かどうかの判定をするツールが、現鈴木研究室の馬場雄介君により作成されている。念のため、それによる確認も行っており、LL(1) 文法であることを確認してある。

6.1 日本語読み数式データの構文解析

日本語読み数式音声データを LL(1) 構文解析する。

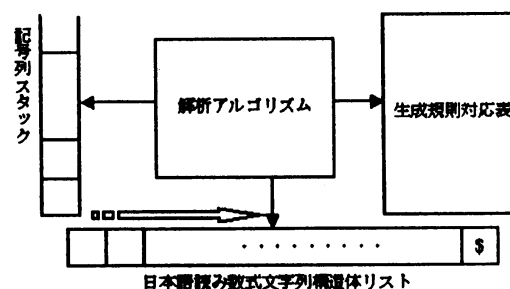


図 1: 日本語読み数式データの構文解析

● 生成規則対応表

終端記号によって、生成規則が一意的に決まる。各終端記号に対して、どの生成規則が対応しているかを表にしてある。

● 記号列スタック

生成規則の中で扱っている記号が、スタックの状態、格納されている。スタックのトップには、あらかじめ、開始記号の Math が入っている。

● 解析アルゴリズム

リストの先頭から、生成規則対応表によって、どの生成規則を用いるか決定する。

6.2 日本語数字読み上げの認識

今回、数字の読み上げは、万までを想定している。一の位より高い数字のときは、その位の数字とその位の桁数である零の数を掛け算する。最後に、それぞれの位の値を足し算して数字が確定する。

小数点に関しては、連続する数字の羅列読みを行ったとき、つまり、一の位が連続して読み上げられたときに、小数以下の数として、数字をならべる。

7 日本語読み数式音声入力インタフェース開発

日本語読み数式音声入力の可能なインタフェースを構築する。単独の音声入力ツールとしては利用しづらいので、InfyEditor アプリケーション⁷に組み込み、そのダイアログ上で、動くようにする。

InfyEditor 上で、日本語読み数式音声入力を行うと、その編集画面上に、数式が表示される。

⁷InfyEditor は、鈴木研究室で開発された数式文書編集機能インタフェースである。InfyEditor の特徴として、テキスト部との併用入力が行えると同時に、TeX に準拠したアルファベットによるコマンド入力、リアルタイムに数式を追加、修正できる点である。TeX などでは面倒であるマージン調整、印刷機能を備えており、テスト問題作成などに厄介な、空白を簡単に入れることができる。操作性に関しては、システムメモリを効率よく利用し、数式入力に関係しない機能を極力取り除くことで、エディタ並みの快適さを実現している。

